HW2

2020147555 컴퓨터과학과 강종서

1. read 메소드

read 메소드는 별도의 lock 이 필요하지 않습니다. 하나의 쓰레드에서 참조하고 있는 table 의 값이 다른 쓰레드에 의해 변할 때 문제가 되는 경우는 우리가 찾고자 하는 값이 table 에서 삭제되는 경우입니다. 하지만 이번 과제에서는 remove 메소드를 호출하는 경우가 없기 때문에 read 메소드 실행과정 전체에 대해 lock을 걸 필요가 없습니다. locked_hash_table.h의 read 메소드에서 lock 에 해당하는 부분을 삭제해주었습니다.

2. insert 메소드

locked_hash_table.h의 insert 메소드의 문제점은 쓰레드 별로 서로 다른 table 위치를 참조하고 있지만 table 전체에 lock이 걸려있어 불필요한 대기 시간이 늘어난다는 점입니다. Mutex 의 개수를 늘려 각각의 mutex 가 범위를 나누어 table 을 관리할 수 있도록 해주었습니다.

```
class better_locked_probing_hash_table : public hash_table {

  private:
    Bucket* table;
    const int TABLE_SIZE; //we do not consider resizing. Thus th
    std::mutex global_mutex;

    /* TODO: put your own code here  (if you need something)*/
    /*****************/
    static const int NUM_OF_MUTEX=5000;
    std::mutex m[NUM_OF_MUTEX];

    /*****************/
    /* TODO: put your own code here */
```

기존 insert 메소드의 while 반복문의 구조를 변경하였습니다. iteration 마다 참조하고자 하는 index에 따라 새롭게 lock을 걸고 다음 iteration으로 넘어가기 전에 unlock을 하도록 변경해주었습니다.

```
int count=0, index=this->hash(key), mutexid;
while(1){
  mutexid=index%NUM_OF_MUTEX;
  m[mutexid].lock();
  if(!table[index].valid){
    mutexid=index%NUM_OF_MUTEX;
    table[index].key=key;
    table[index].valid=true;
    table[index].value=value;
    m[mutexid].unlock();
    return true;
  }

  if(key==table[index].key){
    m[mutexid].unlock();
    return true;
  }
  m[mutexid].unlock();
  count++;
  index=this->hash_next(key,index);
  if(count>=TABLE_SIZE){
    return false;
  }
```

3. 실행 결과

초기에는 쓰레드의 수만큼 mutex 가 있으면 좋겠다고 생각을 해 16개의 mutex 로 실행을 해보았습니다. Init process 0.5초, test process 1.7초의 수행 시간이 걸렸습니다. 다시 생각해보니 mutex 가 많으면 많을 수록 좋을 것 같아 개수를 계속 늘려가며 실행해보았습니다. mutex의 개수가 5000개 일 때 init process 0.12초, test process 0.36초가 나왔고 이 이후로는 개수를 아무리 늘려도 실행 속도가 비슷했습니다.

Mutex 가 1개일 경우도 실행해보았는데 locked_hash_table.h 에서 read 메소드의 lock 을 제거한 코드와 비슷한 속도를 보였습니다.

NUM_OF_MUTEX = 5000 일 때의 실행결과

```
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$ make run
g++ -std=c++11 -g -fopenmp -pthread  main.cc -o HTtest -lboost_system  -lboost_thread
./HTtest 0 | tee result/base.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 0
baseline HT 0
start filling
init hash table took 4.35776 sec
start test
test 36000000 ops took 12.7653 sec
sanity check PASSED:
./HTtest 1 | tee result/better.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 1
better HT 1
start filling
init hash table took 0.119533 sec
start test
test 36000000 ops took 0.372607 sec
sanity check PASSED:
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$
```

NUM_OF_MUTEX = 16 일 때의 실행결과

```
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$ make run
g++ -std=c++11 -g -fopenmp -pthread  main.cc -o HTtest -lboost_system  -lboost_thread
./HTtest 0 | tee result/base.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 0
baseline HT 0
start filling
init hash table took 3.6885 sec
start test
test 36000000 ops took 12.2532 sec
sanity check PASSED:
./HTtest 1 | tee result/better.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 1
better HT 1
start filling
init hash table took 0.510173 sec
start test
test 36000000 ops took 1.67164 sec
sanity check PASSED:
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$
```

NUM_OF_MUTEX = 1 일 때의 실행결과

```
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$ make run
g++ -std=c++11 -g -fopenmp -pthread  main.cc -o HTtest -lboost_system  -lboost_thread
./HTtest 0 | tee result/base.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 0
baseline HT 0
start filling
init hash table took 3.70456 sec
start test
test 36000000 ops took 12.6929 sec
sanity check PASSED:
./HTtest 1 | tee result/better.txt
TABLE_SIZE 10000000 init: 4000000 new: 4000000 NT: 16 additional_reads: 9 use_custom: 1
better HT 1
start filling
init hash table took 3.05532 sec
start test
test 36000000 ops took 4.99219 sec
sanity check PASSED:
mgp2023_20@workspace-ulaz1ky75sg8-0:~/HW2$
```